A knowledge centric approach to conceptualizing robotic solutions

Subhrojyoti Roy Chaudhuri
TCS Research & Innovations,
Tata Consultancy Services Ltd. ,
Pune, Maharashtra, India
subhrojyoti.rc@tcs.com

Amar Banerjee
TCS Research & Innovations,
Tata Consultancy Services Ltd. ,
Pune, Maharashtra, India
amar.banerjee@tcs.com

N. Swaminathan
TCS Research & Innovations,
Tata Consultancy Services Ltd. ,
Chennai, Tamil Nadu,
swami.n@tcs.com

Venkatesh Choppella
Software Engineering Research Centre,
International Institute of Information Technology,
Hyderabad, Telangana, India
venkatesh.choppella@iiit.ac.in

Arpan Pal
TCS Research & Innovations,
Tata Consultancy Services Ltd. ,
Kolkata, West Bengal, India
arpan.pal@tcs.com

Balamurali P.
TCS Research & Innovations,
Tata Consultancy Services Ltd. ,
Bengaluru, Karnataka, India
balamurali.p@tcs.com

ABSTRACT

The focus of the ongoing Digital and Industry 4.0 revolution is on re-engineering business operations to take advantage of various technologies, including robotics. Conceptualizing, say, a robotics solution to automate aspects of warehouse operations involves multiple activities: understanding the problem space in sufficient detail to identify the right automation opportunities; working through the space of possible solution options and developing the solution design; and building a prototype solution with sufficient functional detail to enable customer experts to assess its suitability with respect to multiple concerns, and its impact on the business processes and environment. Only after that can a project be initiated to engineer and deploy the production solution, while making the necessary changes to the business system.

With current practice, conceptualization and prototyping typically takes several months and considerable manual effort. In this paper, we present an environment for rapid prototyping of robotics solutions that facilitates a knowledge-centric approach based on capability composition. The environment enables systematic capture of functional domain knowledge, modular composition of solution space capabilities, and expression of the solution concept using a constrained natural language. Detailed functional simulators are generated automatically from the resulting design. This results in high customer confidence in the solution, substantial reductions in cycle time, and productivity gains due to modular reusability of solution knowledge and components.

ISEC'19, February 14–16, 2019, Pune, India

ACM ISBN 978-1-4503-6215-3/19/02…$15.00
https://doi.org/10.1145/3299771.3299782

CCS CONCEPTS

• Robotics Engineering • Model Driven Engineering • Knowledge Based Engineering• Capability Driven Development • Domain Specific Languages

KEYWORDS

Component Capability, Goal Achievement Process, Model Driven Engineering, Domain Specific Engineering Environment

ACM Reference format:

Subhrojyoti Roy Chaudhuri, Amar Banerjee, N. Swaminathan, Venkatesh Choppella, Arpan Pal and Balamurali P. 2019. A knowledge centric approach to conceptualizing robotic solutions. In *Proceedings of ACM Innovations in Software Engineering Conference (ISEC'19). ACM, Pune, Maharashtra, India, 11 pages.* https://doi.org/10.1145/3299771.3299782

1 Motivation

With Industry 4.0, the world is moving rapidly towards adopting various Digital technologies, including AI and robotics, to automate and transform their business processes. Opportunities for such automation exist in every industrial application domain, including manufacturing, retail, pharma, utilities, process industries etc. Service/consulting firms such as ours engage with multiple customers in each of these domains to identify the right automation opportunities, design appropriate solutions, implement and deploy the solutions with associated business transformations.

Since these solutions involve changes to business operations, there is considerable risk involved in conceptualizing the solution. A solution may be extremely attractive at the high level, but even small

gaps in detailed solution capabilities may cause significant business disruptions. For example, a warehouse robotics solution to fetch items from warehouse shelves may function very well in demo environments, but spillages may result in a slippery factory floor that can cause damage both to goods and the robot itself. Conceptualization should include the creation of proof-of-concept prototypes that exercise the complete operational concept and enable exploration of a wide range of scenarios, so that engineers and customers can work out the complete detailed target business operational state and gain confidence in the solution.

In current practice, solution engineers put in considerable effort to understand the problem environment, work with customers to identify the right transformation opportunities, and work through the range of solution options available to select and design the robotics platform and solution concept. A solution prototype is typically implemented on the selected robotics platform, and exercised extensively to refine the solution concept. This whole approach takes considerable time and effort, usually in the form of a project lasting several months with a team of 5-10 robotics engineers, plus time from customers and domain experts. Cost and effort are increased by the fact that every refinement must be implemented on the actual hardware robotics platform. Moreover, this entire activity must be repeated afresh for each automation solution and each customer, even in the same application domain. A recent internal study on design thinking revealed the challenges involved in this process of problem understanding and solution conceptualization.

Rapid evolution of the robotics field, resulting in an explosion of solution options and available capabilities, further increases the challenges in conceptualizing the right solution. It is clear that there is a need for an alternative approach that can systematically leverage knowledge both of application domains and the solution space (robotics), and enable rapid prototyping and exploration of the conceptualized solutions.

This paper presents such an environment. Solution components at each level of the hierarchy expose capabilities, which are composed using a control system domain-specific language (DSL) to deliver higher-level capabilities. Reusable knowledge in the application and robotics domains is captured using ontological structures. Business workflows are expressed using another DSL. A constrained natural language (CNL) expresses how robotics capabilities are used to perform workflow tasks, thereby making the solution concept accessible to customer experts.

All these elements together constitute a high-level specification of the complete problem and solution, from which we generate simulations of the proposed automation solution, to facilitate refinement. Once the solution has been finalized, implementation code is automatically generated from the model, integrated with available solution components and deployed on the target hardware platforms. This end-to-end support for the engineering life cycle significantly reduces cycle time and increases productivity, particularly as the base of reusable knowledge and components builds up.

Section 2 reviews the current state of the practice with respect to the robotics solution engineering life cycle. Section 3 provides an overview of our solution approach. Section 4 discusses our environment for engineering robotics solutions in detail. Section 5 presents a case study of the usage of the environment, including a comparison with a more traditional engineering approach. Section 6 concludes the paper with a discussion of the contribution and planned further work.

2 State of Art

Current approaches in building robotic solutions extend from modelling high level robotic behavior to software and frameworks that facilitate low level control of the hardware capabilities of the robotics platform. However, there is no end-to-end environment which enables engineers to develop robotic solutions from the top most level of identifying the desired robotic behavior to implementing it via software code. This whole solution engineering process can be broadly classified into five steps [1].

2.1 Understanding the nature of the problem

The first step towards developing a robotic solution is to understand 'what' the robot should do. This understanding is primarily developed by analyzing the nature of the problem, customer interactions and field visits. These activities enable engineers to understand current business operations, identify opportunities for automation, and determine the various capabilities that the robotic solution should provide. It involves understanding the problem from multiple domain viewpoints, including mechatronics, electronics, software and the specific application domain such as manufacturing or warehouse operations. The results are typically captured using text based documents, and requirements that identify the specific capabilities that the robot should be able to deliver. Advanced practices include converting text based requirement documents into more formal formats [2], which facilitate more structured understanding of the problem, and perhaps in terms of robotics domain concepts.

2.2 Solution Space Exploration & Design

Once the robotic problem is understood well enough to identify necessary capabilities that the robotic solution should provide, extensive research is carried out to gather knowledge about various applicable robotics platforms, algorithms, resources, components, and architecture designs which could be potentially be used as a part of the solution. This information gathering activity generates knowledge of the candidate solution space, from within which solution designers must select the options that best fit the problem characteristics.

The candidate solution space consists of a variety of choices at different levels of granularity with different tradeoffs, including both off-the-shelf customizable platforms, and individual components and algorithms from which solutions can be synthesized. Alternative choices may be able to deliver similar functional capabilities, but with different detailed behaviors and quality characteristics.

Solutions designers require deep understanding of the application domain, the specific problem context as well as the robotics domain, in order to make good choices and arrive at candidate designs. Among the candidate designs, engineers choose one or more for prototype development. Further filtration of the solution design is made based on detailed exploration of prototype behavior.

The state of the art supports this design approach with environments such as RobotML [3] and Synergy[4] that provide model-driven engineering (MDE) capabilities to aid designers in developing the robotic solution design. However, while the environments support solution modeling, the choices that drive the content of the solution are left entirely to the knowledge of the engineers gathered by doing the research.

2.3 Creating prototypes and their validation

This stage involves converting the design solutions into operable robotic artefacts, and determining the suitability of the resulting prototypes to address all the concerns and scenarios arising in business operations. Hence it requires a thorough understanding of both robotics and the application domain. As suggested by Elverum et. al. [5], the prototype implements the solution design ideas to test if the ideas are viable, in terms of handling all the desired variety and delivering the targeted quality characteristics. Validation of the prototype to determine its viability is again a highly manual and knowledge-intensive process. Various test case scenarios must be identified, and test environments must be developed which exercise the prototype accordingly.

In current practice, the development of the prototype is typically done by procuring actual hardware and developing the software around it, which consumes a considerable amount of effort and time. Alternatively, the state of the art provides simulation approaches as in GAZEBO [6] and RViz[7], which allow validation of the robotic solution design in computer based simulated environments. Such methodologies prove to be very useful to verify the design without using the actual hardware or real world objects, however it still requires considerable knowledge about the application domain to create the simulated environments. Even if the solution is verified in the simulated environment, it is likely that a prototype, will be produced on the actual hardware to demonstrate its viability in the actual environment, and build confidence in the solution. After iterations of satisfactory validation results from the developed prototypes, the solution is ready for actual implementation. As development of multiple prototypes and their validation is time, effort and cost consuming, generally the number of built prototypes is restricted to one (or a simulation followed by actual hardware testing), and multiple iterations are done over the same prototype along with the necessary design changes.

2.4 Final Development & Testing

After successful prototyping, the final solution is developed and again tested prior to actual implementation of the business operations transformation. This activity becomes very critical as testing is done on the actual robotic hardware, and any unwanted behavior of the application may damage the hardware. Hence the engineers try to minimize this risk by extensive testing of the solution design in the prototyping state itself, so that enough confidence can be gained about the success of the design as a final solution. However, there still exist chances that a design flaw which went un-noticed because of incomplete coverage of the test scenarios, brings about an unwanted behavior into the final robotic application.

From the study of the state of the art, it is evident that, right from understanding the problem domain, to designing solution options, to developing and verifying prototypes and development of the final solution, all the robotic development processes are backed by the various kinds of knowledge. There seems to be no approach that systematically captures this knowledge to apply it during the engineering, to speed up the process of evaluating multiple solutions options and determining preferred solutions. Also, while service/consultancy organizations such as ours work with multiple customers in the same application domain to automate and transform processes, there is no systematic approach to reusing application domain knowledge. Nor is there an integrated life cycle approach and environment, which systematically pulls together all the required information and knowledge in the problem and solution spaces to enable integrated solution engineering and validation. Such an approach would enable solution designers to capture application domain knowledge and information about business operations, explore multiple solutions using various combinations of the available knowledge, select the best robotics solution, create test scenarios, enable rapid prototyping and then implementation of the final solution. It would provide high confidence in the final developed solution. The rest of this paper describes such an environment.

3 Proposed Approach

The challenges in the state of the art serve as our initial requirements. Based on the understanding of the challenges at every state of the robotic development process, we identified four major solution areas that would overcome the challenges:

- Knowledge capture and reuse mechanisms that reduce the time and dependency on external researching to gather various robotics domain knowledge (robotic capabilities knowledge, software libraries, hardware devices, algorithms etc.) as well as application domain knowledge (manufacturing plants, warehouse order retrieval process)
- Enable rapid solution design by rapid composition of existing components, based on a capability composition paradigm.
- Facilitate solution design exploration by enabling engineers to create and evaluate multiple design solutions, reducing the prototyping time
- Enable mechanisms to reduce the cost of building prototypes for the potential solution options via auto generation of simulated environments. Procurement of robotics infrastructure for final validation of the prototype is only needed after solution refinement has already been done on the simulation.

- Enable rapid prototype validation in simulated environments by automating the generation of test scenarios as well.

Based on the above solution areas, we propose a robotics solution engineering platform in form of a domain specific engineering environment (DSEP). The DSEP architecture is discussed in detail in the below section.

3.1 Solution Architecture

The DSEP implements the following solution elements:

- Capture knowledge from the robotics domain about the various re-usable robotic components (hardware devices, software libraries, robotic capabilities like vision, sensing, pick etc.), utilize them to perform application domain tasks, and store these composition procedures back as higher-order capabilities
- Provide an interface to specify task recipes in form of a controlled natural language(CNL), which is close to English and integrates application domain vocabularies, for ease of describing task recipes. Since these are close to natural language, it is easier for customer experts to review these recipes and suggest refinements and scenarios.
- Synthesize design of a robotic solution by orchestrating robotics capabilities based on the specified task recipes, by mapping them to a robotics control implementation.
- Generate robot simulators from the design, along with environment simulators that can be used to validate the solution and explore scenarios.
- Generate actual implementation code from the final solution.

The DSEP incorporates the application domain vocabulary by plugging in the application domain knowledge, which is captured using an ontology. Further this domain vocabulary is used to express the application domain process or the task recipe. The DSEP also plugs into it the robotic domain knowledge which is again stored as ontology. This ontology stores knowledge about the various robotic components like software libraries, robotic hardware devices (Picker Arm, Vision Camera etc.) and capabilities which are provided by these components.

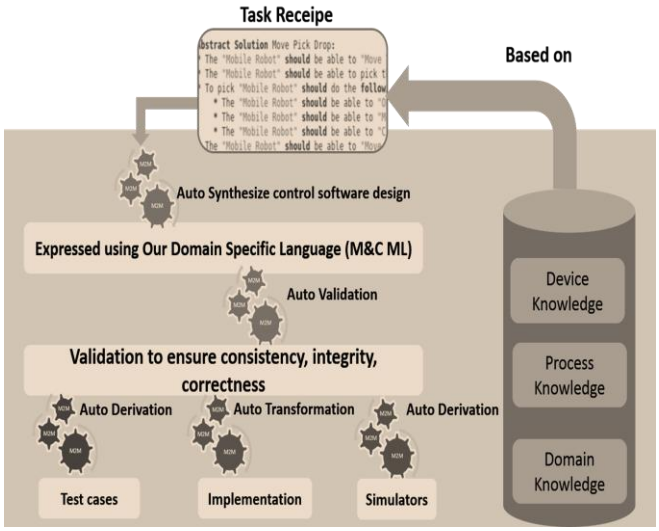

Figure 1: DSEP architecture

The user specifies the task recipe, along with its mapping to robotics capabilities from the robotics knowledge, and the application context of the robotic solution from the application domain. This mapping between the task recipe, capability and the context details is synthesized to generate the solution design, the implementation code and the simulation configuration for the solution. The architecture provides an end to end approach to robotic solution development so that multiple solution options could be explored using rapid prototyping.

4 Implementation Details

The following sub sections discuss the primary solution implementation components. Section 4.1 and 4.2 discuss the nature of the domain knowledge along with its capture details. Section 4.3 discusses implementation details for the CNL, which is used to specify the task recipes. The synthesis of a solution design and generation of implementation and simulation configurations for rapid prototyping is discussed in section 4.4.

4.1 Application domain knowledge

DSEP incorporates the application domain knowledge which is captured externally using OWL[8]. **Figure 2**, provides a snapshot of the application domain knowledge.

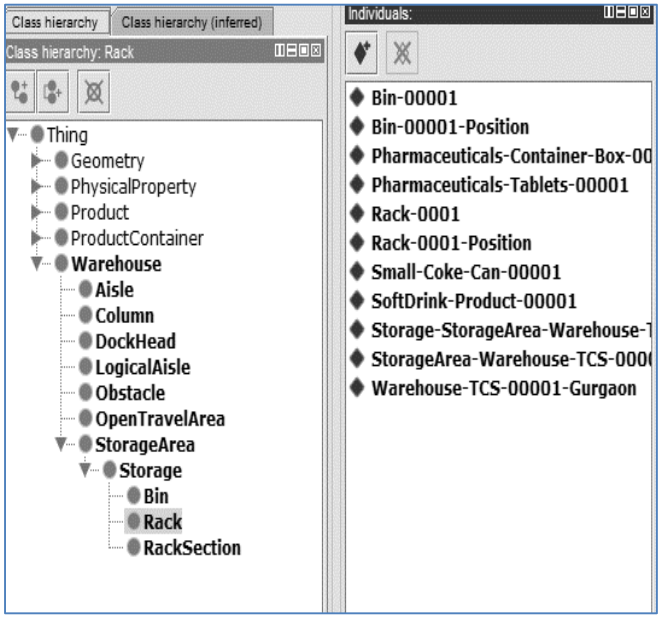

Figure 2: Application Domain Knowledge Ontology

The application domain knowledge is captured by identifying the concepts in the application domain knowledge as ontology classes. For example, A Warehouse domain contains concepts like Warehouse, Rack, Products, Aisle, Storage Area etc. These concepts in the warehouse domain have relation with the Physics & Geometry domain in form of a physical and geometrical properties like mass,

volume, width, shape etc. Such relations are captured in the ontology classes while capturing the application domain knowledge. The concepts in the application domain are further instantiated as individuals for an instance of the warehouse.

4.2 Robotics knowledge

Experts from the robotics domain possess implicit knowledge about the resources required to develop robotic solutions. This also requires extensive software development knowledge such as programming platforms, libraries utilized for software implementation and so on. Further, it requires knowledge about simulation models of the individual resources to implement prototypes in a simulated software environment without requiring procurement of the actual hardware resources.

Our goal is to reduce the time and effort involved in having dependencies on extensive research, to gather the knowledge, to create a robotic solution design. Our approach enables cataloguing of such implicit knowledge in terms of capabilities from the robotics domain, again in an ontology form. The ontology captures knowledge about the various hardware devices / robotic components and their simulation details, software libraries in form of robotic capability knowledge. We use the capability driven approach to capture the robotic domain knowledge, so that the knowledge about what a robotic component (hardware / software) is capable of perform can be stored as knowledge. This knowledge is then utilized while specifying the task recipes as discusses in section 4.4.

Inspired by Bērziša et al [9], we define capability as the ability and capacity that enable a resource to achieve a functional goal in a certain context. We separate the capability of a robotic component into two separate aspects a) functional description and b) their behavior logic.

Based on our exploration of the current state of art, we identified that functional description of capabilities can be captured using the SRDL[10] meta-model implemented by KnowRob[11], which provides a meta-model to identify relation between capabilities, components and robot structure. The structure of the SRDL meta-model is shown in the below.

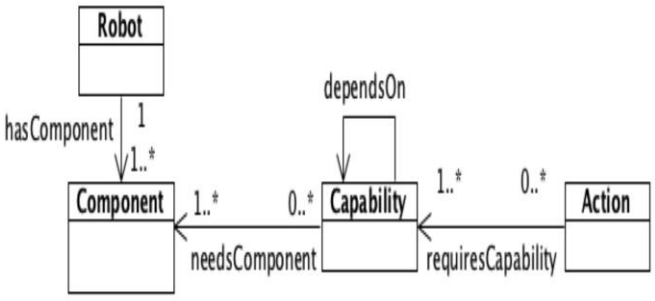

Figure 3: SRDL Model in form of a class diagram

As seen in **Figure 3**, the SRDL model provides a reference structure of the robot model by linking capabilities and components. However, using the SRDL model, behavior details of resources such as how a resource needs to be instantiated and instructed (example by sending specific commands), handled for errors, states and their transition logic and so on, cannot be described. To mitigate this, we model the resource behavior using the Monitoring and Control Modelling Language (M&C ML) DSL[12], which we developed as a part our earlier work related to control systems. However, the modelling of the resource behavior is not enough, as it still lacks the mapping to a specific capability like Picking, Moving, Sensing etc. This mapping is established by creating an ontology for the robotics knowledge where capabilities are mapped with resources and their behaviors. The sections below discuss capability modeling by leveraging SRDL as well as M&C ML

4.2.1 Resource knowledge and their Behavior

Resource knowledge provides the required guidance towards the resource usage. For example, it captures details such as the right sequence of instructions accepted by the resource to invoke and use a capability, its errors handling strategies and so on. We term this aspect as the behavior knowledge of the resource and capture it using M&C ML DSL. The main purpose of this DSL is to model the design of a control and monitoring software. Since the problem of coordinating a collection of components (with associated capabilities) to deliver a desired higher level capability is basically a control systems problems, it is appropriate to use a control systems DSL for this purpose. **Figure 4** shows usage of M&C ML to capture the resource behavior.

```
        InterfaceDescription Navigate {
            commands {
                move[],stop[],left[],right[]
            }
            events{
                reached_destination[]
            }
            alarms{
                navigation_failed[]
            }
            operatingStates{
                stopped[], moving[], turning[]
            }
        }
ControlNode Nav_CN {
Associated Interface Description : Navigate
CommandResponseBlock {
    Command Navigator.Navigate.move {
        Transitions {
            currentState Navigator.Navigate.stopped =>
            nextState Navigator.Navigate.moving
        }
    }
}
AlarmBlock {
    Alarm Navigator.Navigate.navigation_failed{
        AlarmHandling {
            Action [
                fireCommands :Navigator.Navigate.stop
            ]
        }
```

Figure 4: Capturing resource behavior using M&C ML

M&C ML provides the vocabulary to define the interface of a resource (software / hardware), in terms of commands, events, alarms, operating states and data points. Control Node block models the behavior of the resource to implementing a specific capability. This block allows modeling the resource behavior for commands, alarms, events, data points and rules for state transitions. The result of the block is a higher-level capability to perform a particular application task such as grasping an item or fetching an item.

Business operations are typically described in terms of activities with associated workflows. These workflows ultimately decompose the activity into individual tasks that are performed by robotic actors and components. From the viewpoint of specifying compositional logic, workflow specifications are theoretically equivalent in expressive power to control systems specifications. However, since business operations logic is typically specified as sequenced workflows rather than state-machine-oriented control logic, we created a separate DSL for this purpose, Capability Workflow Language (CWL), which facilitates expression of composite capabilities, and is more friendly to business users. An overview of this can be understood from **Figure 5.**

In CWL, activities are recursively broken down into lower-level tasks till they can be mapped with suitable capability from the resource-capability knowledge, for their implementation.

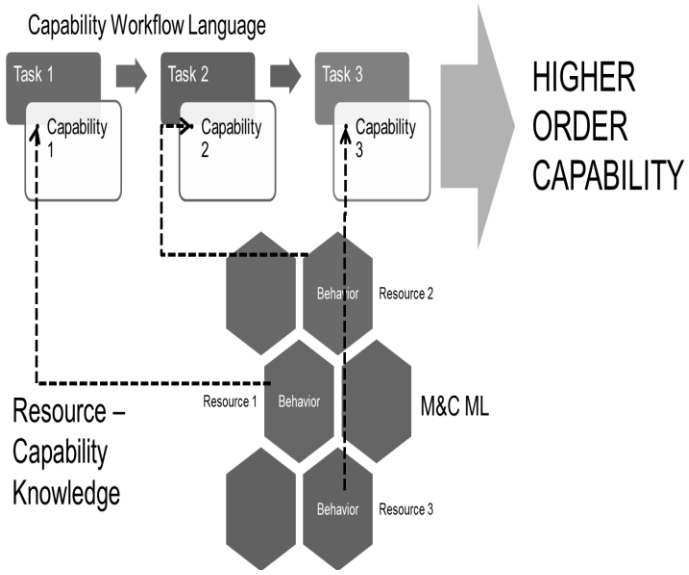

Figure 5: Activity description using DSL and binding capability

Although it was possible to use modeling languages such as UML [13], WebWorkFlow [14] and SysML[15] to capture such details, we developed a CWL DSL (aligned with the meta-model of SysML) that enables activity flow to be specified textually. The key motivation behind this was to enable the CWL to be specified in terms of the capabilities already modeled using M&C ML. Since M&C ML was developed using XText [16] and EMF[17], our implementation of the activity flow language enabled it to easily integrate with M&C ML. The nature of the CWL is shown in **Figure 6.**

```
ActivityDiagram Navigate has activities
{
    Activity Turn_Left {
    requireCapability (Navigator.Navigate.left)
    nextActivity : Move_Straight
    },
    Activity Move_Straight {
    requireCapability (Navigator.Navigate.move)
    nextActivity: Turn_Right
    },
    Activity Turn_Right {
    requireCapability (Navigator.Navigate.right)
    }
```

Figure 6: Creating activities using CWL

4.2.2 Functional description of Capability

The capabilities are categorized by three OWL classes namely perception, navigation and action in the OWL ontology, as shown in **Figure 7**, The OWL classes establish the relation of the capability classes with the resources providing these capabilities and the resource behavior which achieves this capability. Also the resource-behavior relation for capability classes also provide placeholders to capture knowledge about the simulation details like simulation model, simulation configuration specifications etc.

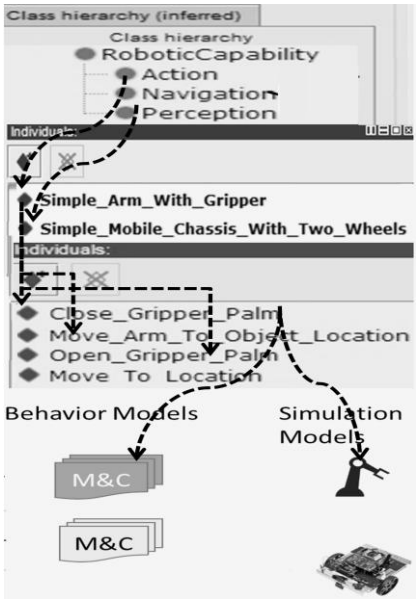

Figure 7: Robot Capability Ontology

Individuals of these abstract classes capture the functional description of the capabilities. The knowledge also contains mapping of the functional descriptions with the knowledge about the corresponding off-the-shelf resources that implement them. This whole structured knowledge ontology along wit

4.3 Controlled Natural Language

A Controlled Natural Language (CNL)[18], defined by us as part of DSEP enables the application domain knowledge and the robotics capability to be pulled together to specify a task recipe describing a domain process for a robot based solution. The language allows specification of robotic capabilities for their utilization for implementing the process. The grammar of the CNL is of the following form:

A ROBOT (Collection of resources modelled using M&C ML) +
Executes ***A CAPABILITY*** (Capability provided by the resources as stored in the robotics domain knowledge) +
Involving ***an OBJECT*** (Captured knowledge about the application domain)

Examples of robotic capabilities are *move to location, open palm, close palm* etc. and examples of domain vocabulary are terms such as *Rack, Aisle, Can, Product and so on.* Hence combing these two knowledge bases, the CNL allows specifying a statement such as *move to location of the Rack.*

The CNL implements a combination of technologies used for developing domain specific languages (DSL's) and Natural Language Processing (NLP). The CNL allows specifying conditional statements and loops as well. **Figure 8** provides an example of the CNL usage snapshot.

```
Abstract Solution Move Pick Drop:
• The "Mobile Robot" should be able to "Move To Location" of "Rack".
• The "Mobile Robot" should be able to pick the "SmallCan".
• To pick "Mobile Robot" should do the following:-
    * The "Mobile Robot" should be able to "Open Gripper Palm" on "Bin".
    * The "Mobile Robot" should be able to "Move Arm"  to the location of "SmallCan".
    * The "Mobile Robot" should be able to "Close Gripper Palm" and hold the "SmallCan". end
• The "Mobile Robot" should be able to "Move To Location" of "Bin".
• The "Mobile Robot" shall drop the "SmallCan" in the 'Bin'.
• To drop "Mobile Robot" should do the following:-
    * The "Mobile Robot" should be able to "Move Arm" to the target location in the "Bin".
    * The "Mobile Robot" should be able to "Open Gripper Palm" and drop the "SmallCan". end
```

Figure 8: A Task recipe to Move, Pick Object and Drop in Bin using CNL

4.4 Auto Synthesis of Simulator

One of our main goals was to provide an approach that enables rapid prototyping based on the high level task description created using the CNL. The synthesis takes the Task recipe as the input and generates the prototype implementation along with the simulation configuration so validate the solution design specified. Here the solution design is abstracted at a high level in form of the Task recipe specified using the CNL. The task recipe incorporates the resources which would be a part of the solution, their behavior, capabilities and the step by step workflow which needs to be executed in the solution. Hence this itself becomes a solution design. This solution design is then utilized to realize rapid prototyping by auto generating the prototype implementation and its simulation specification to validate it's working.

4.4.1 Unstructured to structured task description

In this step, the task description specified using the CNL converts into a structured task description, expressed using the CWG. The algorithm for it is:

```
Generate_Structured_Task_Description(Task_Receipe)
{
INITIALIZE activityFlow correspondingTo Task_Recipe
FOR ALL task IN Task_Receipe)
        INITIALIZE activity correspondingTo task
        SET     Task_capability                 =
        task.hasTaskCapability
        SET       application_domain_object     =
        task.targetObject
        SET      activity.requiredCapability    =
        Task_capability
        SET          activity.inputData         =
        application_domain_object.dataModel
        IF (task has nextTask)
                Activity.hasNextActivity        =
                task.nextTask
        IF (task has sub-taskRecipe)
                SET   activity.childActivityFlow  =
                Generate_Structured_Task_Description
                (task.sub-taskReceipeS)
        ADD activity in activityFlow
RETURN activityFlow
}
```

4.4.2 Synthesis of a control system design

The tasks described in the CWL contains the necessary inputs to generate the control system design. In this stage, the control system design expressed again using M&C ML, auto generates based on processing the activities in the CWL and the resource-capability knowledge in M&C ML. The algorithm used to synthesize the control system design is shown below:

```
.
Create_Control_Design (activityFlow)
{
INITIAIZE Controller corresponding to activityFlow
FOR ALL activities IN activityFlow
        INITIAIZE Operating_State corresponding to
        activity
        ADD  Operating_State  to  the  Controller
        interface
        IF activity has nextActivity
                INITIALIZE  State_Transition  from
                initial  state  corresponding  to
                activity to the next state corre-
                sponding to next activity
                ADD  State  _Transition  to  the
                Controller
        IF activity requires Capability
                SET
                Controller_implementing_Capability =
                Controller     which     implements
                Capability
                SET   Controller.childController  =
                Controller_implementing_Capability
```

```
            SET Capability as the action call for
            State_Transition
        ELSE      IF      activity      requires
        Executable_operation_Script
            SET  Executab1e_operation_Script  as
            the action call for State_Transition
        ELSE IF activity requires childActivityFlow
            Create_Control_Design
            (childActivityFlow)
RETURN Controller
}
```

The algorithm to strategize the task execution flow is given below.

```
Generate_Task_Strategy (Control_Design)
{
DECLARE Task_Strategy
FOR Controller IN Control_Design.hasControllers
        SET   Task_Execution_Capability   =   Get
        Capability from Controller responsible for
        task execution
        ADD      Task_Execution_Capability      TO
        Task_Strategy
        FOR Child_Controller IN Control-
        ler.hasChildControllers
            SET Child_Task_Execution_capability
            Generate       Task       Strategy
            (Child_Controller)
            ADD  Child_Task_Execution_capability
            TO Task Execution Capability
RETURN Task_Strategy}
```

The generated control system design contains controllers for underlying resources providing the lower-order capabilities and supervisory logic to coordinate them.

4.4.3 Automated Code Generation for simulation

The generated control system design models all the lower devices, their capabilities and their coordination logic. As this design is implemented using MDE, we also exploit MDE further to generate the executable code for the design including the configuration to simulate it. We retarget the task described as an Activity Flow into multiple implementation artefacts in ROS [19] and GAZEBO. The generated software is a control system and implements a task execution strategy to coordinate the execution of the capabilities as a state machine following the structures of SMACH [20] model.

This code is deployed along with the simulation models for the resources captured as knowledge as per SDF format [21]. This allows the entire generated code to be execute in GAZEBO by utilizing the application domain knowledge example warehouse and the software controllers. The pseudo code to generate the simulation environment from the activityFlow (CWL) is described below. The simulation environment generation gathers data from the activity flow generated from the CNL and fathers the corresponding simulation details from the application knowledge ontology, by querying the knowledge ontology model.

```
Generate_Environment (activityFlow)
{
SET context_environment = activityDi-
agram.context_details
DECLARE environment_gazebo_model
FOR ALL element IN context_environment.elements
        SET      element_structure      =      get
        element_structure from ontology
        SET element_gazebo_model = generate element
        model format (element_structure)
        ADD element_gazebo_model IN environ-
        ment_gazebo_model
RETURN environment_gazebo_model
}
```

5 Use case

To validate this approach against a real life scenario, a case study was conducted. We collaborated with one of our groups who had already developed software for a pick and place robotic system as part of their participation in a robotics challenge. We collected relevant data about the robot controller software they implemented manually, as shown in Table 1.

Table1. Software engineering artefacts for manual implementation of usecase

Parameters	Value
Lines of code	1500
No. of packages	1
No. of files	21
Reused libraries	9
Coding structure	Non-modular
Time taken to develop the software	30 days

Through this exercise, we wanted to check what fraction of the solution could be modeled as capability knowledge and to what extent the approach could generate the solution automatically. We faced the following challenges:

- The code written by the team was not modularized and properly structured
- The reusable libraries and specific code were intertwined in the same package

As a first step towards using our approach, we re-organized their code in the following manner. This consumed about a couple weeks of effort.

The main script containing the entry point function was identified. This script was then re-factored to contain bare minimum high level logic which is executed to run the robotic tasks (SMACH script). The business logic encoded in the main script was segregated into different libraries. These libraries were then packaged by following a naming convention as per the executing tasks in the main script. This restructuring allowed us to model the required capabilities, resources and soft-ware libraries accordingly. We saw that 66% of the code was possible to be generated automatically as they could be modeled as robotics capabilities knowledge and the remaining 34% of the code had specific business logic which needed to be manually written.

5.1 Our approach to address the usecase

Following our proposed architecture, we approached this use case by capturing knowledge by implementing the following steps:

1. Capture the context knowledge as application domain knowledge using OWL ontology
   - Capture all items in the application context as individuals in the OWL ontology
   - Capture knowledge relating to the physical shape and geometry about the objects in the context
   - Capture SDF based simulation models of these objects in the context as
2. Capture device knowledge in terms of the resource interface and behavior using M&C ML DSL
   - Capture the resource interface model using M&C ML
   - Capture resource behavior model using M&C ML
   - Capture the resource capabilities knowledge as robotic capability ontology and map the robotic capabilities with the interface and behavior models
   - Capture the resource simulation details as simulation models, and simulation configuration for the resources in the ontology.
3. Create a task recipe to pick an object from a rack and drip it in the box using the application domain and robotic capability knowledge
4. Auto generate activity flow description from the CNL.
5. Auto generate the prototype implementation and its corresponding simulation configuration for GAZEBO simulator

To capture the simulation details for the robotic resources like the Kuka [22] Arm (Robotic Arm), the objects in the context like the Rack, Bin etc. re-used these models from the GAZEBO models repository. Likewise, the robotic capability knowledge can be incrementally enriched with more and more resources with their capability and simulation details. Similarly based on the application domain, the simulation models corresponding to various application domains like Warehouse, Manufacturing Unit etc. can be stored in the application domain knowledge ontology. The nature of these types of knowledge can be seen in Figure: 2,3 and 4.

After generating the prototype implementation and the simulation configuration we were able to generate the orchestration logic for the design prototype in ROS framework and GAZEBO SDF format respectively. The orchestrator generated for the task execution for the Robotic Arm is shown in Figure:8

```
                                    input_keys  =['movetoobjectapi'],
                                    result_cb=movetoobject_action_response.execute,
output_keys =[]),
transitions={'preempted':'preempted' ,'aborted':'aborted' ,'succeeded':'GRASP' },
remapping={}
)

smach.StateMachine.add('GRASP', SimpleActionState('Gripper_Move_Server', GripperMoveAction,
                                    goal_cb=grasp_action_request.execute,
                                    input_keys  =['pickapi'],
                                    result_cb=grasp_action_response.execute,
output_keys =[]),
transitions={'preempted':'preempted' ,'aborted':'aborted' ,'succeeded':'PLACE_PALLET' },
remapping={}
)

smach.StateMachine.add('PLACE_PALLET', SimpleActionState('Arm_Move_Server', ArmMoveAction,
                                    goal_cb=movetobin_action_request.execute,
                                    input_keys  =['movetobinapi'],
                                    result_cb=movetobin_action_response.execute,
output_keys =[]),
transitions={'preempted':'preempted' ,'aborted':'aborted' ,'succeeded':'DROP' },
remapping={}
)

smach.StateMachine.add('DROP', SimpleActionState('Gripper_Move_Server', GripperMoveAction,
                                    goal_cb=drop_action_request.execute,
                                    input_keys  =['dropapi'],
                                    result_cb=drop_action_response.execute,
output_keys =[]),
transitions={'preempted':'preempted' ,'aborted':'aborted' ,'succeeded':'succeeded' },
remapping={}
```

Figure 9: Generated SMACH script to orchestrate robotic arm

As seen in Figure 8, this approach aims towards capturing knowledge about the device, its capabilities, the application context and much of the implicit details generates into the code. This knowledge then can be re-used for a different robotic solution. Using this approach, we generate all the standard executable blocks which form a part of the orchestration logic for the robotic solution. As a result, all the ROS prescribed Actions, Messages, Services [19] etc. were auto-generated which usually consume a lot of human effort to create. We also generated the simulation environment for GAZEBO simulator to execute the solution prototype to validate its execution. The simulation environment was generated as SDF format XML file. The simulation spec. for the context is shown in **Fig. 10** as an XML tree visualization. This SDF world simulation environment was generated by populating all the context objects from the task recipe and generating corresponding models.

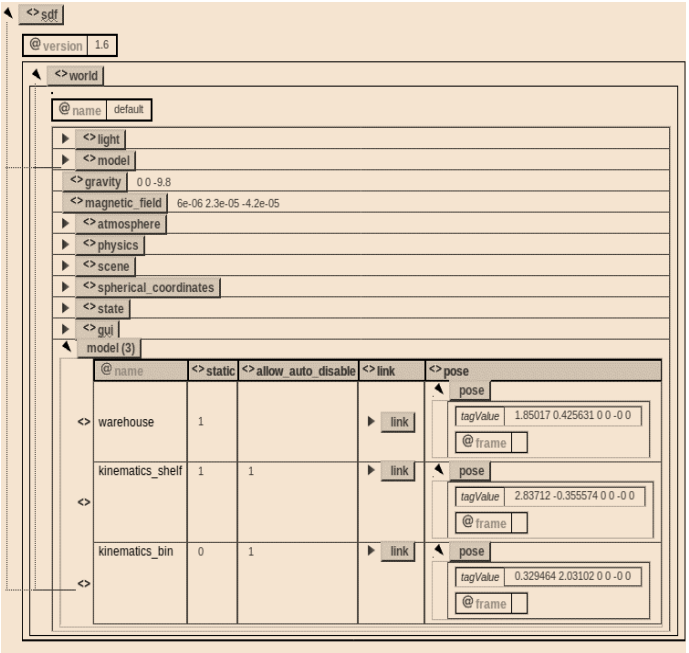

Figure 10: Simulation SDF model

The SDF model corresponding to the exact robotic components specified in the task description may not be available. Since open source tools like GAZEBO have limited set of predefined models, incorporating exhaustive set of robot models as domain knowledge is a challenge in this approach.

The approach also generated the Kuka Arm(Robot) model, which was stored as knowledge ontology corresponding to the Kuka resource. The Figure 10, shows the Kuka arm executing the task recipe in a simulated environment, generated from the DSEP. **Figure 10** shows a screenshot of the execution of the solution prototype in a simulated environment.

Figure 11: Execution of prototype in simulated environment

Through this approach, multiple solution design options can be explored by altering the resource-capability mapping in the task recipe. Multiple such task recipes could be created and the prototype could be quickly generated. This prototype could be then evolved by minimal manual efforts to specify specific logic as per the engineer's understanding. This effort also reduces the effort to manually develop simulation environment and scenarios, by generating the simulation configurations according to the knowledge captured.

Once the solution has been refined using the prototype, the environment can be used to generate actual implementation code for the particular robotics platform, by changing the code generator bindings appropriately. This has the advantage that the final implementation is less likely to have deviations from the prototype that was thoroughly validated.

This use case showed an effort reduction of approximately 60% (2.5X productivity) and the end-to-end solution engineering cycle time was cut approximately in half (2X), though precise comparisons were difficult due to various factors. If there is a considerable base of reusable knowledge and components (as would be the case after multiple usages of the environment), then we believe that the cycle time and productivity gains would be even higher. In addition, the quality of the final solution would also be better, because the knowledge base facilitates a more exhaustive exploration of solution options.

6 Conclusion

Industry 4.0 involves automation and transformation of business processes using robotics and other Digital technologies. There is considerable risk in conceptualizing the transformed business system, since even minor solution gaps and omitted scenarios can lead to significant business disruption. Rapid prototyping and simulation are essential to enable solution refinement. But the current approach of creating *ad hoc* prototypes for each customer and each solution is effort-intensive and results in long cycle times for technology adoption.

We have developed an engineering environment that enables a modular, compositional approach to solution conceptualization and specification. It enables generation of both simulations and production implementations from the model, improving both productivity and cycle time significantly. While this environment is specifically aimed at robotics solutions, the underlying principles of capability composition, domain knowledge capture and model-driven generation of simulations and implementations, are all applicable to other Digital solutions as well.

While we currently have a preliminary implementation of the environment, there is considerable room for expansion and refinement of its capabilities. We also plan to formalize the concepts involved in capability composition, and develop principles and tools for capability matching between problem and solution spaces.

REFERENCES

[1] W. Nowak, U. Reiser and E. Prassler, 2012. Robot Development Process in the DESIRE Project. In Towards Service Robots for Everyday Environments (pp. 507-516). Springer, Berlin, Heidelberg.

[2] J.O. Ringert, B. Rumpe and A. Wortmann, 2014. A Requirements Modeling Language for the Component Behavior of Cyber Physical Robotics Systems. arXiv preprint arXiv:1409.0394.

[3] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi and M. Ziane, 2012, November. Robotml, a domain-specific language to design, simulate and deploy robotic applications. In International Conference on Simulation, Modeling, and Programming for Autonomous Robots (pp. 149-160). Springer, Berlin, Heidelberg.

[4] L.K. Katragadda, 1997. Synergy: a language and framework for robot design (Doctoral dissertation, Carnegie Mellon University).

[5] C.W. Elverum, T. Welo and S. Tronvoll, 2016. Prototyping in new product development: Strategy considerations. Procedia CIRP, 50, pp.117-122.

[6] N.P. Koenig and A. Howard, 2004, September. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In IROS (Vol. 4, pp. 2149-2154).

[7] D. Hershberger, D. Gossow and J. Faust, RViz, 3D visualization tool for ROS. URL: http://wiki. ros. org/rviz [cited 06-08-2018].

[8] D.L. McGuinness and F. Van Harmelen, 2004. OWL web ontology language overview. W3C recommendation, 10(10), p.2004.

[9] S. Bērziša, G. Bravos, T.C. Gonzalez, U. Czubayko, S. España, J. Grabis, M. Henkel, L. Jokste, J. Kampars, H. Koç and J.C. Kuhr, 2015. Capability driven development: an approach to designing digital enterprises. Business & Information Systems Engineering, 57(1), pp.15-25.

[10] L. Kunze, T. Roehm and M. Beetz, 2011, May. Towards semantic robot description languages. In Robotics and Automation (ICRA), 2011 IEEE International Conference on (pp. 5589-5595). IEEE.

[11] M. Tenorth and M. Beetz, 2009, October. KnowRob—knowledge processing for autonomous personal robots. In Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on (pp. 4261-4266). IEEE.

[12] P. Patwari, S.R. Chaudhuri, S. Natarajan and G. Muralikrishna, 2016. M&C ML: A modeling language for monitoring and control systems. Fusion Engineering and Design, 112, pp.761-765.

[13] B. Selic, 1998. Using UML for modeling complex real-time systems. In Languages, compilers, and tools for embedded systems (pp. 250-260). Springer, Berlin, Heidelberg.

[14] Z. Hemel, R. Verhaaf and E. Visser, 2008, September. WebWorkFlow: an object-oriented workflow modeling language for web applications. In International Conference on Model Driven Engineering Languages and Systems (pp. 113-127). Springer, Berlin, Heidelberg.

[15] S. Friedenthal, A. Moore and R. Steiner, 2014. A practical guide to SysML: the systems modeling language. Morgan Kaufmann.

[16] L. Bettini, 2016. Implementing domain-specific languages with Xtext and Xtend. Packt Publishing Ltd.

[17] D. Steinberg, F. Budinsky, E. Merks and M. Paternostro, 2008. EMF: eclipse modeling framework. Pearson Education.

[18] B. Davis, C.M. Keet and A. Wyner eds., 2018. Controlled Natural Language: Proceedings of the Sixth International Workshop, CNL 2018, Maynooth, Co. Kildare, Ireland, August 27-28, 2018 (Vol. 304). IOS Press.

[19] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A.Y. Ng, 2009, May. ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

[20] J. Bohren and S. Cousins, 2010. The SMACH high-level executive [ROS news]. IEEE Robotics & Automation Magazine, 17(4), pp.18-20.

[21] OSRF. SDF. 2014 Retrieved August, 2018 from: http://sdformat.org/spec

[22] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald and G. Hirzinger, 2010, June. The KUKA-DLR Lightweight Robot arm-a new reference platform for robotics research and manufacturing. In Robotics (ISR), 2010 41st international symposium on and 2010 6th German conference on robotics (ROBOTIK) (pp. 1-8). VDE.

[23] K.C. Kang, M. Kim, J. Lee and B. Kim, 2005, September. Feature-oriented re-engineering of legacy systems into product line assets–a case study. In International Conference on Software Product Lines (pp. 45-56). Springer, Berlin, Heidelberg.

[24] R.R. Yeddula, S. Vale, S. Reddy, C.P. Malhotra, B.P. Gautham and P. Zagade, 2016. A Knowledge Modeling Framework for Computational Materials Engineering. In SEKE (pp. 197-202).

[25] C. Schlegel, A. Steck, D. Brugali and A. Knoll, 2010, November. Design abstraction and processes in robotics: From code-driven to model-driven engineering. In International Conference on Simulation, Modeling, and Programming for Autonomous Robots (pp. 324-335). Springer, Berlin, Heidelberg.

[26] C. Schlegel, A. Steck and A. Lotz, 2012. Robotic software systems: From code-driven to model-driven software development. In Robotic Systems-Applications, Control and Programming. InTech.

[27] A. Diego, V.C. Cristina, O. Francisco, P. Juan and Á. Bárbara, 2010. V3cmm: A 3-view component meta-model for model-driven robotic software development. Journal of Software Engineering in Robotics, 1(1), pp.3-17.

[28] A. Bubeck, F. Weisshardt and A. Verl, 2014, June. Bride-a toolchain for framework-independent development of industrial service robot applications. In ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of (pp. 1-6). VDE.

[29] M. Tenorth and M. Beetz, 2012, March. Knowledge processing for autonomous robot control. In AAAI Spring Symposium: Designing Intelligent Robots.